Celem pierwszego projektu z zaawansowanych języków programowania jest dokonania refaktoryzacji poniższego kodu napisanego w jawie, stworzonego podczas ćwiczenia GildedRose .Klasa Item opisuje strukturę typowego przedmiotu zaś klasa GildedRose(numer przy klasie oznacza iteracje kodu czyli 0 oznacza podstawową wersję bez zmian) zawiera listę przedmiotów oraz funkcje operujące na tej liście. Celem najważniejszej funkcji updateQuality jest aktualizowanie w zbiorze przedmiotów ich wartości sellIn i quality w zależności od typu przedmiotu.

```
package gildedRose;

public class Item {

    public String name;
    public int sellIn;
    public int quality;

    public Item(String name, int sellIn, int quality) {
     this.name = name;
     this.sellIn = sellIn;
     this.quality = quality;
    }

    @Override
    public String toString() {
      return this.name + ", " + this.sellIn + ", " + this.quality;
    }
}
```

```
package gildedRose;

public class GildedRose_0 {
       Item[] items;

          public GildedRose_0(Item[] items) {
              this.items = items;
          }

          public void updateQuality() {
              for (int i = 0; i < items.length; i++) {
                  if (!items[i].name.equals("Aged Brie")
                          && !items[i].name.equals("Backstage passes to a
TAFKAL80ETC concert")) {
                      if (items[i].quality > 0) {
                          if (!items[i].name.equals("Sulfuras, Hand of
Ragnaros")) {
                             if(!items[i].name.contains("Conjured")){
                               items[i].quality = items[i].quality - 1;
                             }else{
                               if(items[i].quality <2 ){
                                   items[i].quality = 0;
                               } else {items[i].quality = items[i].quality - 2;
                               }
                             }
                          }
                      }
                  } else {
                      if (items[i].quality < 50) {
                          items[i].quality = items[i].quality + 1;
```

```
                        if (items[i].name.equals("Backstage passes to a
TAFKAL80ETC concert")) {
                            if (items[i].sellIn < 11) {
                                if (items[i].quality < 50) {
                                    items[i].quality = items[i].quality + 1;
                                }
                            }

                            if (items[i].sellIn < 6) {
                                if (items[i].quality < 50) {
                                    items[i].quality = items[i].quality + 1;
                                }
                            }
                        }
                    }
                }

                if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                    items[i].sellIn = items[i].sellIn - 1;
                }

                if (items[i].sellIn < 0) {
                    if (!items[i].name.equals("Aged Brie")) {
                        if (!items[i].name.equals("Backstage passes to a
TAFKAL80ETC concert")) {
                            if (items[i].quality > 0) {
                                if (!items[i].name.equals("Sulfuras, Hand of
Ragnaros")) {

                                    if(!items[i].name.contains("Conjured")){
                                      items[i].quality = items[i].quality -
1;

                                    }
                                    else{
                                       if(items[i].quality <2
){items[i].quality = 0;

                                       } else{items[i].quality =
items[i].quality - 2;

                                       }
                                    }


                                }
                            }
                        } else {
                            items[i].quality = items[i].quality -
items[i].quality;
                        }
                    } else {
                        if (items[i].quality < 50) {
                            items[i].quality = items[i].quality + 1;
                        }
                    }
                }
            }
        }

}
```

Funkcja updateQuality działa, jednak jest ona bardzo długa, posiada ona bardzo długie wielokrotnie zagnieżdżone instrukcje warunkowe oraz niepotrzebne powtórzenia przez co jest ona mało czytelna zaś dodawanie nowych typów przedmiotów jest trudniejsze z każdym nowym typem. W celu zwiększenia czytelności tego kodu trzeba zastosować jego refaktoryzacje. Jedną z rzeczy jaką można zauważyć w kodzie są wielokrotne instrukcje warunkowe sprawdzające czy wartość zmiennej quality należy do przedziału <0,50> by móc je potem usunąć w ramach refaktoryzacji tworzymy w klasie GildedRose funkcję legaliseQuality o poniższym kodzie.

```
public void legaliseItemQuality(Item item) {
  if(item.quality > 50) item.quality = 50;
  if(item.quality < 0) item.quality = 0;
}
```

Funkcja ta sprawdza czy wartość quality należy do przedziału ,jeśli nie to ustawia jej wartość na 0 lub 50 w zależności czy jest ona dodatnia czy ujemna.

Następnym krokiem jest modyfikacja funkcji updateQuality. Zaczynamy modyfikacje od poniższego fragmentu kodu.

```
      } else {
              if (items[i].quality < 50) {
                items[i].quality = items[i].quality + 1;

                if (items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                  if (items[i].sellIn < 11) {
                    if (items[i].quality < 50) {
                      items[i].quality = items[i].quality + 1;
                    }
                  }

                  if (items[i].sellIn < 6) {
                    if (items[i].quality < 50) {
                      items[i].quality = items[i].quality + 1;
                    }
                  }
                }
              }

            }
```

Fragment ten określa zmiany w zmiennej value dla przedmiotów typu BackStagePass i AgedBrie. Zaczynamy modyfikacje od stworzenia funkcji która określi jak zmienić wartość przedmiotu typu BackStagePass w zależności od sellIn

```
public int backstagePassssSellInModifier(int sellIn) {
  int modifier = 0 ;
   if (sellIn  < 11) modifier++;
   if (sellIn  < 6) modifier++;
    return modifier;
}
```

Ta funkcja pozwala zastąpić kod zaznaczony na czerwono zaś funkcja legaliseItemQuality kod zaznaczony na niebiesko w wyniku czego otrzymujemy kod poniżej.

```
} else {
  items[i].quality = items[i].quality + 1;
    if (items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                  items[i].quality = items[i].quality +
backstagePassssSellInModifier(items[i].sellIn);
    }
```

```
        legaliseItemQuality(items[i]);
}
```

W następnym kroku modyfikujemy poniższy fragment kodu który określa zmiany w zmiennej quality gdy przedmiot jest domyślnego typu lub typu Conjured.

```
if (!items[i].name.equals("Aged Brie")
                        && !items[i].name.equals("Backstage passes to a
TAFKAL80ETC concert")) {
                    if (items[i].quality > 0) {
                        if (!items[i].name.equals("Sulfuras, Hand of
Ragnaros")) {
                            if(!items[i].name.contains("Conjured")){
                              items[i].quality = items[i].quality - 1;
                            }else{
                              if(items[i].quality <2 ){
                                  items[i].quality = 0;
                              } else {items[i].quality = items[i].quality - 2;
                              }
                            }
                        }
                    }
                } else {
```

Tak jak poprzedniej modyfikacji korzystamy z funkcji legaliseItemQuality by pozbyć się sprawdzania wartości zmiennej quality(kod oznaczony na niebiesko) w wyniku czego otrzymujemy kod poniżej.

```
         if (!items[i].name.equals("Aged Brie")
                    && !items[i].name.equals("Backstage passes to a TAFKAL80ETC
concert")) {

                    if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                        if(!items[i].name.contains("Conjured")){
                          items[i].quality = items[i].quality - 1;
                        }else{
                          items[i].quality = items[i].quality - 2;

                        }
                    }
                    legaliseItemQuality(items[i]);
            } else {
```

Dotychczasowy kod funkcji updateQuality wygląda następująco.

```
    public void updateQuality() {
        for (int i = 0; i < items.length; i++) {
            if (!items[i].name.equals("Aged Brie")
                    && !items[i].name.equals("Backstage passes to a TAFKAL80ETC
concert")) {

                    if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                        if(!items[i].name.contains("Conjured")){
                          items[i].quality = items[i].quality - 1;
                        }else{
                          items[i].quality = items[i].quality - 2;

                        }
                    }
                    legaliseItemQuality(items[i]);
```

```
            } else {

                    items[i].quality = items[i].quality + 1;

                    if (items[i].name.equals("Backstage passes to a TAFKAL80ETC
concert")) {
                      items[i].quality = items[i].quality +
backstagePassssSellInModifier(items[i].sellIn);
                    }
                    legaliseItemQuality(items[i]);
            }

            if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                items[i].sellIn = items[i].sellIn - 1;
            }

            if (items[i].sellIn < 0) {
                if (!items[i].name.equals("Aged Brie")) {
                    if (!items[i].name.equals("Backstage passes to a
TAFKAL80ETC concert")) {
                        if (items[i].quality > 0) {
                            if (!items[i].name.equals("Sulfuras, Hand of
Ragnaros")) {
                                if(!items[i].name.contains("Conjured")){
                                  items[i].quality = items[i].quality - 1;
                                }
                                else{
                                 if(items[i].quality <2 ){items[i].quality = 0;
                                 } else{items[i].quality = items[i].quality -2;
                                    }
                                }

                            }
                        }
                    } else {
                        items[i].quality = items[i].quality - items[i].quality;
                    }
                } else {
                    if (items[i].quality < 50) {
                        items[i].quality = items[i].quality + 1;
                    }
                }
            }
        }
    }
```

Dotychczasowe zmiany miały za zadanie zmniejszyć ilość instrukcji warunkowych w kodzie głównie przy pomocy funkcji legaliseQuality. Kolejnym krokiem jest usunięcie wszystkich instrukcji warunkowych sprawdzających wartość zmiennej quality w funkcji updateQuality. Dotychczas funkcja legaliseQuality była wstawiana na koniec każdego zmienianego fragmentu kodu by funkcja updateQuality działała dalej poprawnie jednak tym kroku będzie można już usunąć te powtórzenia i zastąpić je jednym jej wystąpieniem na koniec pętli iterującej po przedmiotach . Funkcja ta musi znajdować wewnątrz bloku warunkowego sprawdzającego czy przedmiot nie jest typu Sulfuras gdyż wartość value nie może być zmieniana dla tego typu przedmiotu. Kod który modyfikujemy w tym kroku został oznaczony tak jak poprzednio przy pomocy koloru niebieskiego. Po zastosowaniu wyżej opisanych zmian finalnie otrzymany kod ma postać

```
public void updateQuality() {
        for (int i = 0; i < items.length; i++) {
            if (!items[i].name.equals("Aged Brie")
                    && !items[i].name.equals("Backstage passes to a TAFKAL80ETC
concert")) {

                    if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                        if(!items[i].name.contains("Conjured")){
                          items[i].quality = items[i].quality - 1;
                        }else{
                          items[i].quality = items[i].quality - 2;

                        }
                    }
            } else {

                    items[i].quality = items[i].quality + 1;

                    if (items[i].name.equals("Backstage passes to a TAFKAL80ETC
concert")) {
                      items[i].quality = items[i].quality +
backstagePassssSellInModifier(items[i].sellIn);
                    }
            }

            if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                items[i].sellIn = items[i].sellIn - 1;
            }

            if (items[i].sellIn < 0) {
                if (!items[i].name.equals("Aged Brie")) {
                    if (!items[i].name.equals("Backstage passes to a
TAFKAL80ETC concert")) {
                                if (!items[i].name.equals("Sulfuras, Hand of
Ragnaros")) {
                                    if(!items[i].name.contains("Conjured")){
                                      items[i].quality = items[i].quality - 1;
                                    }
                                    else{
                                      items[i].quality = items[i].quality - 2;
                                    }

                                }
                    } else {
                        items[i].quality = items[i].quality - items[i].quality;
                    }
                } else {

                        items[i].quality = items[i].quality + 1;
                }
            }

             if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
             legaliseItemQuality(items[i]);

             }
        }
}
```

Po spojrzeniu na powyższy kod można zauważyć że to jak zmienia się wartość zmiennej quality zależy od tego jaki mamy typ przedmiotu oraz czy wartość zmiennej sellIn jest mniejsza od 0 lub większa-równa zero. Wyjątkami są przedmioty typu Sulfuras których wartości zmiennych są niezmienne. Zależność tą można wyrazić za pomocą pojedynczej funkcji.

```
public void qualityModifier(Item item,int modifier1,int modifier2) {
  item.quality = (item.sellIn < 0) ? item.quality+modifier1 :
item.quality+modifier2;
}
```

Funkcja ta dodaje wartość modifier1 do wartości zmiennej value jeśli wartość sellIn jest mniejsza od 0 lub modifier2 w przeciwnym wypadku.

Kolejnym krokiem w refaktoryzacji jest użycie powyższej funkcji by stworzyć funkcje dla każdego typu przedmiotu opisujące jak zmienią się ich wartości zmiennej value dla konkretnych wartości sellIn w tym konkretnym typie przedmiotu.

Zaczynamy od zmienienia domyślnej wartości modyfikatora w funkcji backstagePassSellInModifier z 0 na 1 ponieważ typ BackStagePass nie dzieli dłużej kodu z przedmiotem typu AgedBrie. Po tej zmianie funkcja ta wygląda następująco.

```
public int backstagePassssSellInModifier(int sellIn) {
   int modifier = 1 ;
   if (sellIn  < 11) modifier++;
   if (sellIn  < 6) modifier++;
   return modifier;
}
```

Po zmianie tej funkcji tworzymy funkcje obsługującą zmianę wartości value dla przedmiotu typu backstagePass na podstawie kodu oznaczonego na ciemno-czerwono(kod wspólny z typem AgedBrie) i niebiesko(Kod dotyczycący tylko typu BackStagePass) i otrzymujemy poniższą funkcję .

```
public void backstagePassesHandler(Item pass) {
qualityModifier(pass,-pass.quality,backstagePassssSellInModifier(pass.sellIn));
}
```

Tworzymy funkcje obsługującą zmianę wartości value dla przedmiotu typu AgedBrie na podstawie kodu oznaczone go na ciemno-czerwono i czerwono (Kod dotyczycący tylko typu AgedBrie) i otrzymujemy poniższą fukcję .

```
public void agedBrieHandler(Item brie) {
    qualityModifier(brie,2,1);
}
```

Tworzymy funkcje obsługującą zmianę wartości value dla przedmiotu typu Conjured na podstawie kodu oznaczonego na różowo(Kod dotyczycący tylko typu Conjured) otrzymujemy poniższą fukcję .

```
public void conjuredItemHandler(Item conjuredItem) {
    qualityModifier(conjuredItem,-4,-2);
}
```

Tworzymy funkcje obsługującą zmianę wartości value dla przedmiotu typu Default na podstawie kodu oznaczonego na ciemno-zielono(Kod dotyczycący tylko typu Default) i otrzymujemy poniższą funkcję .

```
public void defaultItemHandler(Item defaultItem) {
    qualityModifier(defaultItem,-2,-1);
}
```

Przy pomocy tych funkcji dokonujemy kolejnej refaktoryzacji kodu klasy GildedRose do postaci poniżej.

```
public class GildedRose_1 {
       Item[] items;

          public GildedRose_1(Item[] items) {
              this.items = items;
          }
          public void legaliseItemQuality(Item item) {
            if(item.quality > 50) item.quality = 50;
            if(item.quality < 0) item.quality = 0;
          }

          public int backstagePassssSellInModifier(int sellIn) {
           int modifier = 1 ;
           if (sellIn  < 11) modifier++;
              if (sellIn  < 6) modifier++;
           return modifier;
          }

           public void qualityModifier(Item item,int modifier1,int modifier2) {
             item.quality = (item.sellIn < 0) ? item.quality+modifier1 :
             item.quality+modifier2;
          }

          public void backstagePassesHandler(Item pass) {
             qualityModifier(pass,0,backstagePassssSellInModifier(pass.sellIn));
          }
          public void agedBrieHandler(Item brie) {
             qualityModifier(brie,2,1);
          }

          public void conjuredItemHandler(Item conjuredItem) {
            qualityModifier(conjuredItem,-4,-2);
          }

          public void defaultItemHandler(Item defaultItem) {
            qualityModifier(defaultItem,-2,-1);
          }

          public void updateQuality() {
              for (int i = 0; i < items.length; i++) {
                 if(!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                   if (items[i].name.equals("AgedBrie"))
                     agedBrieHandler(items[i]);
                    else if(items[i].name.equals("Backstage passes to a
                     TAFKAL80ETC concert"))backstagePassesHandler(items[i]);
                    else if(items[i].name.contains("Conjured"))
                    conjuredItemHandler(items[i]);
                    else defaultItemHandler(items[i]);
                    legaliseItemQuality(items[i]);
                    items[i].sellIn--;
                 }
              }
          }
}
```

Jak widać powyżej zmiany wartości zmiennej value w przedmiotach innych niż Sulfuras są teraz obsługiwane przez odpowiadające im funkcje. W funkcji updateQuality sprawdzamy czy przedmiot nie jest typu Sulfuras by nie zmienić wartości jego zmiennych sellIN i value, jeśli nie to w zależności o typu przedmiotu wywołujemy odpowiadającą im funkcje potem sprawdzamy czy wartość quality jest w dopuszczalnym zakresie <0,50> i na koniec zmniejszamy wartość sellIn o 1.Kod wygląda znacznie lepiej niż na początku. Został on znacząco skrócony a zmiany w przedmiotach zostały oddelegowane do konkretnych funkcji jednak dodawanie nowych rodzajów przedmiotów wymaga modyfikacji funkcji updateQuality przez dodawanie nowych instrukcji warunkowych co sprawia że kod tej funkcji staje się coraz bardziej nieczytelny z każdym nowym przedmiotem. Innym problem jest fakt że różne rodzaje przedmiotów posiadają specjalne cechy je opisujące co wymaga dodatkowych instrukcji warunkowych np. trzeba sprawdzać czy przedmiot jest typu Sulfuras by nie zmienić wartości jego zmiennych. By rozwiązać powyższe problemy w następnym etapie stworzymy klasy dla każdego typu przedmiotu i przeniesiemy do nich funkcje opisujące dany typ usuwając przedmiot z listy argumentów tych funkcji i zmodyfikowaniu nazw zmiennych by odnosiły się do zmiennych wewnątrz klasy(np. zamieniamy item.quality na quality ponieważ odnosimy się do zmiennej klasy w której teraz znajduje się funkcja).

Zaczynamy od przeniesienia funkcji legaliseItemQuality, qualityModifier oraz defaultItemHandler której nazwę zmieniamy na dailyUpdate do klasy Item. Oprócz tego można zauważyć dla większości typów przedmiotów zmniejszamy sellIn o 1 jeden i sprawdzamy czy wartość quality należy do przedziału <0,50> dlatego tworzymy funkcję w tej klasie o nazwie legaliseAndDecSellIn która łączy te funkcjonalności i następnie dodajemy ją na koniec funkcji dailyUpdate. Ostatecznie klasa Item ma kod jak poniżej.

```
public class Item {

    public String name;
    public int sellIn;
    public int quality;

    public Item(String name, int sellIn, int quality) {
     this.name = name;
     this.sellIn = sellIn;
     this.quality = quality;
    }

    @Override
    public String toString() {
      return this.name + ", " + this.sellIn + ", " + this.quality;
    }

    protected void legaliseItemQuality() {
      if(quality > 50) quality = 50;
      if(quality < 0)  quality = 0;
    }

    protected void legaliseAndDecSellIn(){
      legaliseItemQuality();
      sellIn--;
    }

    protected void qualityModifier(int modifier1,int modifier2) {
      quality = (sellIn < 0) ? quality+modifier1 : quality+modifier2;
    }

    public void dailyUpdate() {
      qualityModifier(-2,-1);
      legaliseAndDecSellIn();
```

```
    }
}
```

W następnym kroku tworzymy klasę BackStagePass dziedziczącą po klasie Item do której przenosimy funkcje backstagePassssSellInModifier i BackStagePassesHandler której nazwę zmieniamy na dailyUpdate(Nazwa ta będzie zmieniania dla wszystkich klas Handler w kolejnych krokach by nadpisywały one funkcje dailyUpdate z klasy bazowej Item).

```
package gildedRose;

public class BackStagePass extends Item {

	public BackStagePass(String name, int sellIn, int quality) {
		super(name, sellIn, quality);
	}

	public int backstagePassssSellInModifier() {
		int modifier = 1;
		if(sellIn < 11) modifier++;
		if(sellIn < 6) modifier++;
		return modifier;
	}

	@Override
	public void dailyUpdate() {
	   qualityModifier(-quality,backstagePassssSellInModifier());
	   legaliseAndDecSellIn();
	}

}
```

Następnie tworzymy klasę AgedBrie dziedziczącą po klasie Item do której przenosimy funkcje AgedBrieHandler.

```
public class AgedBrie extends Item {

	public AgedBrie(String name, int sellIn, int quality) {
		super(name, sellIn, quality);
	}

	@Override
	public void dailyUpdate() {
		qualityModifier(2,1);
		legaliseAndDecSellIn();
	}
}
```

Tworzymy klasę Conjured dziedziczącą po klasie Item do której przenosimy funkcje ConjuredItemHandler.

```
public class Conjured extends Item  {

	public Conjured(String name, int sellIn, int quality) {
		super(name, sellIn, quality);
	}

	@Override
	public void dailyUpdate() {
		qualityModifier(-4,-2);
```

```
            legaliseAndDecSellIn();
        }
}
```

Jako ostatnią tworzymy klasę Sulfuras dziedziczącą po klasie Item z pustą funkcją dailyUpdate.

```
public class Sulfuras extends Item{

      public Sulfuras(String name, int sellIn, int quality) {
            super(name, sellIn, quality);
      }

      @Override
       public void dailyUpdate(){}
}
```

Utworzenie powyższych klas i przeniesienie funkcji pozwala ostatecznie zredukować klase GildedRose to postaci poniżej.

```
public class GildedRose_2 {
       Item[] items;

          public GildedRose_2(Item[] items) {
              this.items = items;
          }

          public void updateQuality() {
            for (Item item : items) {
                        item.dailyUpdate();
                  }
          }
}
```

Jak widać powyżej w refaktoryzacja pozwoliła zredukować funkcję updateQuality do jednej pętli wykonującej jedną funkcję dailyUpdate której działanie jest zależne od klasy przedmiotu.

Statystyki zmian w kodzie miedzy bazową a końcową wersją mogą być częściowo przedstawione za pomocą poniższych metryk.

Mierzona metryka	Domyślny projekt GildedRose	Ostateczny projekt GildedRose
Liczba linii	84	89
Liczba plików	2	6
Najdłuższa funkcja(ilość linii)	62	6
Średnia długość funkcji(ilość linii)	18.25	3.65
Maksymalna złożoność cyklomatyczna	23	3
Liczba klas	2	6
Liczba funkcji	4	17

Jak widać na podstawie danych zawartych w tabeli powyżej refaktoryzacja przyniosła niewiele większą całkowitą liczbę linii kodu i znacznie większą ilość klas i funkcji ale rozbicie projektu na dodatkowe klasy i funkcje pozwoliło na znaczne skrócenie średniej długości funkcji i zmniejszenie złożoności kodu oraz zwiększenie jego czytelności. Przeniesienie funkcjonalności do dodatkowych funkcji pozwoliło pozbyć się zduplikowanych fragmentów kodu zaś rozbicie kodu odnoszącego się do konkretnych typów przedmiotów na dodatkowe klasy pozwoliło na całkowite oddzielenie kodu opisującego te przedmioty od głównej funkcji updateQuality ,dzięki czemu jakiekolwiek zmiany w kodzie przedmiotów nie wymagają modyfikacji kodu tej funkcji zaś dodawanie nowego przedmiotu odbywa się za pomocą dodania nowej klasy dziedziczącej po

klasie Item. dzięki czemu kod opisujący każdy z typów przedmiotów jest czytelniejszy i łatwiejszy w zrozumieniu ponieważ nie jest on przeplatany z kodem innych typów przedmiotów.